

Optimal search trees with 2-way comparisons^{*}

Marek Chrobak^{1**}, Mordecai Golin^{2***}, J. Ian Munro^{3†}, and Neal E. Young¹

¹ University of California — Riverside, Riverside, California, USA

² Hong Kong University of Science and Technology, Hong Kong, China

³ University of Waterloo, Waterloo, Canada

Abstract. In 1971, Knuth gave an $O(n^2)$ -time algorithm for the classic problem of finding an optimal binary search tree. Knuth’s algorithm works only for search trees based on 3-way comparisons, but most modern computers support only 2-way comparisons ($<$, \leq , $=$, \geq , and $>$). Until this paper, the problem of finding an optimal search tree using 2-way comparisons remained open — poly-time algorithms were known only for restricted variants. We solve the general case, giving (i) an $O(n^4)$ -time algorithm and (ii) an $O(n \log n)$ -time additive-3 approximation algorithm. For finding optimal *binary split trees*, we (iii) obtain a linear speedup and (iv) prove some previous work incorrect.

1 Background and statement of results

In 1971, Knuth [10] gave an $O(n^2)$ -time dynamic-programming algorithm for a classic problem: *given a set \mathcal{K} of keys and a probability distribution on queries, find an optimal binary-search tree T* . As shown in Fig. 1, a search in such a tree for a given value v compares v to the root key, then (i) recurses left if v is smaller, (ii) stops if v equals the key, or (iii) recurses right if v is larger, halting at a leaf. The comparisons made in the search must suffice to determine the relation of v to all keys in \mathcal{K} . (Hence, T must have $2|\mathcal{K}| + 1$ leaves.) T is optimal if it has minimum *cost*, defined as the expected number of comparisons assuming the query v is chosen randomly from the specified probability distribution.

Knuth assumed *three-way* comparisons at each node. With the rise of higher-level programming languages, most computers began supporting only two-way comparisons ($<$, \leq , $=$, \geq , $>$). In the 2nd edition of Volume 3 of *The Art of Computer Programming* [11, §6.2.2 ex. 33], Knuth commented

... machines that cannot make three-way comparisons at once... will have to make two comparisons... it may well be best to have a binary tree whose internal nodes specify either an equality test or a less-than test but not both.

But Knuth gave no algorithm to find a tree built from *two-way* comparisons (a 2WCST, as in Fig. 2(a)), and, prior to the current paper, poly-time algorithms

^{*} This is the full version of an extended abstract that appeared in ISAAC [2].

^{**} Research funded by NSF grants CCF-1217314 and CCF-1536026.

^{***} Research funded by HKUST/RGC grant FSGRF14EG28.

[†] Research funded by NSERC and the Canada Research Chairs Programme.

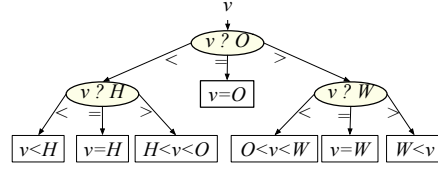


Fig. 1. A binary search tree T using 3-way comparisons, for $\mathcal{K} = \{H, O, W\}$.

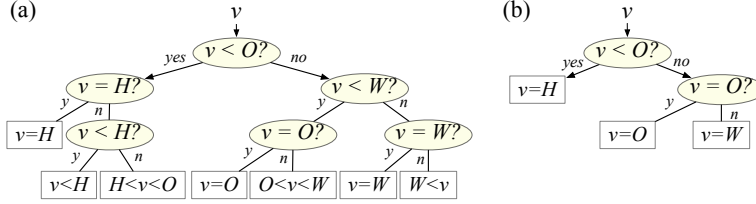


Fig. 2. Two 2WCSTs for $\mathcal{K} = \{H, O, W\}$; tree (b) only handles *successful* queries.

were known only for restricted variants. Most notably, in 2002 Anderson et al. [1] gave an $O(n^4)$ -time algorithm for the *successful-queries* variant of 2WCST, in which each query v must be a key in \mathcal{K} , so only $|\mathcal{K}|$ leaves are needed (Fig. 2(b)). The *standard* problem allows arbitrary queries, so $2|\mathcal{K}| + 1$ leaves are needed (Fig. 2(a)). For the standard problem, no polynomial-time algorithm was previously known. We give one for a more general problem that we call 2WCST:

Theorem 1. *2WCST has an $O(n^4)$ -time algorithm.*

We specify an instance \mathcal{I} of 2WCST as a tuple $\mathcal{I} = (\mathcal{K} = \{K_1, \dots, K_n\}, \mathcal{Q}, \mathcal{C}, \alpha, \beta)$. The set \mathcal{C} of allowed comparison operators can be any subset of $\{<, \leq, =, \geq, >\}$. The set \mathcal{Q} specifies the queries. A solution is an optimal 2WCST T among those using operators in \mathcal{C} and handling all queries in \mathcal{Q} . This definition generalizes both standard 2WCST (let \mathcal{Q} contain each key and a value between each pair of keys), and the successful-queries variant (take $\mathcal{Q} = \mathcal{K}$ and $\alpha \equiv 0$). It further allows any query set \mathcal{Q} between these two extremes, even allowing $\mathcal{K} \not\subseteq \mathcal{Q}$. As usual, β_i is the probability that v equals K_i ; α_i is the probability that v falls between keys K_i and K_{i+1} (except $\alpha_0 = \Pr[v < K_1]$ and $\alpha_n = \Pr[v > K_n]$).⁴

To prove Thm. 1, we prove Spuler’s 1994 “maximum-likelihood” conjecture: *in any optimal 2WCST tree, each equality comparison is to a key in \mathcal{K} of maximum likelihood, given the comparisons so far* [14, §6.4 Conj. 1]. As Spuler observed, the conjecture implies an $O(n^5)$ -time algorithm; we reduce this to $O(n^4)$ using

⁴ As defined here, a 2WCST T must determine the relation of the query v to every key in \mathcal{K} . More generally, one could specify any partition \mathcal{P} of \mathcal{Q} , and only require T to determine, if at all possible using keys in \mathcal{K} , which set $S \in \mathcal{P}$ contains v . For example, if $\mathcal{P} = \{\mathcal{K}, \mathcal{Q} \setminus \mathcal{K}\}$, then T would only need to determine whether $v \in \mathcal{K}$. We note without proof that Thm. 1 extends to this more general formulation.

standard techniques and a new perturbation argument. Anderson et al. proved the conjecture for their special case [1, Cor. 3]. We were unable to extend their proof directly; our proof uses a different local-exchange argument.

We also give a fast additive-3 approximation algorithm:

Theorem 2. *Given any instance $\mathcal{I} = (\mathcal{K}, \mathcal{Q}, \mathcal{C}, \alpha, \beta)$ of 2WCST, one can compute a tree of cost at most the optimum plus 3, in $O(n \log n)$ time.*

Comparable results were known for the successful-queries variant ($\mathcal{Q} = \mathcal{K}$) [16,1]. We approximately reduce the general case to that case.

Binary split trees “split” each 3-way comparison in Knuth’s 3-way-comparison model into two 2-way comparisons within the same node: an equality comparison (which, by definition, must be to the maximum-likelihood key) and a “<” comparison (to any key) [13,3,8,12,6]. The fastest algorithms to find an optimal binary split tree take $O(n^5)$ -time: from 1984 for the successful-queries-only variant ($\mathcal{Q} = \mathcal{K}$) [8]; from 1986 for the standard problem (\mathcal{Q} contains queries in all possible relations to the keys in \mathcal{K}) [6]. We obtain a linear speedup:

Theorem 3. *Given any instance $\mathcal{I} = (\mathcal{K} = \{K_1, \dots, K_n\}, \alpha, \beta)$ of the standard binary-split-tree problem, an optimal tree can be computed in $O(n^4)$ time.*

The proof uses our new perturbation argument (Sec. 3.1) to reduce to the case when all β_i ’s are distinct, then applies a known algorithm [6]. The perturbation argument can also be used to simplify Anderson et al.’s algorithm [1].

Generalized binary split trees (GBSTs) are binary split trees without the maximum-likelihood constraint. Huang and Wong [9] (1984) observe that relaxing this constraint allows cheaper trees — the maximum-likelihood conjecture fails here — and propose an algorithm to find optimal GBSTs. We prove it incorrect!

Theorem 4. *Lemma 4 of [9] is incorrect: there exists an instance — a query distribution β — for which it does not hold, and on which their algorithm fails.*

This flaw also invalidates two algorithms, proposed in Spuler’s thesis [15], that are based on Huang and Wong’s algorithm. We know of no poly-time algorithm to find optimal GBSTs. Of course, optimal 2WCSTs are at least as good.

2WCST without equality tests. Finding an optimal *alphabetical encoding* has several poly-time algorithms: by Gilbert and Moore — $O(n^3)$ time, 1959 [5]; by Hu and Tucker — $O(n \log n)$ time, 1971 [7]; and by Garsia and Wachs — $O(n \log n)$ time but simpler, 1979 [4]. The problem is equivalent to finding an optimal 3-way-comparison search tree when the probability of querying any key is zero ($\beta \equiv 0$) [11, §6.2.2]. It is also equivalent to finding an optimal 2WCST in the successful-queries variant with only “<” comparisons allowed ($\mathcal{C} = \{<\}$, $\mathcal{Q} = \mathcal{K}$) [1, §5.2]. We generalize this observation to prove Thm. 5:

Theorem 5. *Any 2WCST instance $\mathcal{I} = (\mathcal{K} = \{K_1, \dots, K_n\}, \mathcal{Q}, \mathcal{C}, \alpha, \beta)$ where β is not in \mathcal{C} (equality tests are not allowed), can be solved in $O(n \log n)$ time.*

Definitions 1 Fix an arbitrary instance $\mathcal{I} = (\mathcal{K}, \mathcal{Q}, \mathcal{C}, \alpha, \beta)$.

For any node N in any 2WCST T for \mathcal{I} , N 's query subset, \mathcal{Q}_N , contains queries $v \in \mathcal{Q}$ such that the search for v reaches N . The weight $\omega(N)$ of N is the probability that a random query v (from distribution (α, β)) is in \mathcal{Q}_N . The weight $\omega(T')$ of any subtree T' of T is $\omega(N)$ where N is the root of T' .

Let $\langle v < K_i \rangle$ denote an internal node having key K_i and comparison operator $<$ (define $\langle v \leq K_i \rangle$ and $\langle v = K_i \rangle$ similarly). Let $\langle K_i \rangle$ denote the leaf N such that $\mathcal{Q}_N = \{K_i\}$. Abusing notation, $\omega(K_i)$ is a synonym for $\omega(\langle K_i \rangle)$, that is, β_i .

Say T is irreducible if, for every node N with parent N' , $\mathcal{Q}_N \neq \mathcal{Q}_{N'}$.

In the remainder of the paper, we assume that only comparisons in $\{<, \leq, =\}$ are allowed (i.e., $\mathcal{C} \subseteq \{<, \leq, =\}$). This is without loss of generality, as “ $v > K_i$ ” and “ $v \geq K_i$ ” can be replaced, respectively, by “ $v \leq K_i$ ” and “ $v < K_i$ ”.

2 Proof of Spuler's conjecture

Fix any irreducible, optimal 2WCST T for any instance $\mathcal{I} = (\mathcal{K}, \mathcal{Q}, \mathcal{C}, \alpha, \beta)$.

Theorem 6 (Spuler's conjecture). *The key K_a in any equality-comparison node $N = \langle v = K_a \rangle$ is a maximum-likelihood key: $\beta_a = \max_i \{\beta_i : K_i \in \mathcal{Q}_N\}$.*

The theorem will follow easily from Lemma 1:

Lemma 1. *Let internal node $\langle v = K_a \rangle$ be the ancestor of internal node $\langle v = K_z \rangle$. Then $\omega(K_a) \geq \omega(K_z)$. That is, $\beta_a \geq \beta_z$.*

Proof. (Lemma 1) Throughout, “ $\langle v \prec K_i \rangle$ ” denotes a node in T that does an inequality comparison (\leq or $<$, not $=$) to key K_i . Abusing notation, in that context, “ $x \prec K_i$ ” (or “ $x \not\prec K_i$ ”) denotes that x passes (or fails) that comparison.

Assumption 1 (i) All nodes on the path from $\langle v = K_a \rangle$ to $\langle v = K_z \rangle$ do inequality comparisons. (ii) Along the path, some other node $\langle v \prec K_s \rangle$ separates key K_a from K_z : either $K_a \prec K_s$ but $K_z \not\prec K_s$, or $K_z \prec K_s$ but $K_a \not\prec K_s$.

It suffices to prove the lemma assuming (i) and (ii) above. (Indeed, if the lemma holds given (i), then, by transitivity, the lemma holds in general. Given (i), if (ii) doesn't hold, then exchanging the two nodes preserves correctness, changing the cost by $(\omega(K_a) - \omega(K_z)) \times d$ for $d \geq 1$, so $\omega(K_a) \geq \omega(K_z)$ and we are done.)

By Assumption 1, the subtree rooted at $\langle v = K_a \rangle$, call it T' , is as in Fig. 3(a): Let child $\langle v \prec K_b \rangle$, with subtrees T_0 and T_1 , be as in Fig. 3.

Lemma 2. *If $K_a \prec K_b$, then $\omega(K_a) \geq \omega(T_1)$, else $\omega(K_a) \geq \omega(T_0)$.*

(This and subsequent lemmas in this section are proved in Appendix 7.2. The idea behind this one is that correctness is preserved by replacing T' by subtree (b) if $K_a \prec K_b$ or (c) otherwise, implying the lemma by the optimality of T .)

Case 1: Child $\langle v \prec K_b \rangle$ separates K_a from K_z . If $K_a \prec K_b$, then $K_z \not\prec K_b$, so descendant $\langle v = K_z \rangle$ is in T_1 , and, by this and Lemma 2, $\omega(K_a) \geq \omega(T_1) \geq \omega(K_z)$,

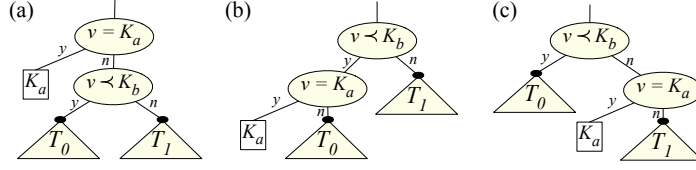


Fig. 3. (a) The subtree T' rooted at $\langle v = K_a \rangle$ and possible replacements (b), (c).

and we're done. Otherwise $K_a \not\prec K_b$, so $K_z \prec K_b$, so descendant $\langle v = K_z \rangle$ is in T_0 , and, by this and Lemma 2, $\omega(K_a) \geq \omega(T_0) \geq \omega(K_z)$, and we're done.

Case 2: *Child $\langle v \prec K_b \rangle$ does not separate K_a from K_z .* Assume also that descendant $\langle v = K_z \rangle$ is in T_1 . (If descendant $\langle v = K_z \rangle$ is in T_0 , the proof is symmetric, exchanging the roles of T_0 and T_1 .) Since descendant $\langle v = K_z \rangle$ is in T_1 , and child $\langle v \prec K_b \rangle$ does not separate K_a from K_z , we have $K_a \not\prec K_b$ and two facts:

Fact A: $\omega(K_a) \geq \omega(T_0)$ (by Lemma 2), and

Fact B: the root of T_1 does an inequality comparison (by Assumption 1).

By Fact B, subtree T' rooted at $\langle v = K_a \rangle$ is as in Fig. 4(a):

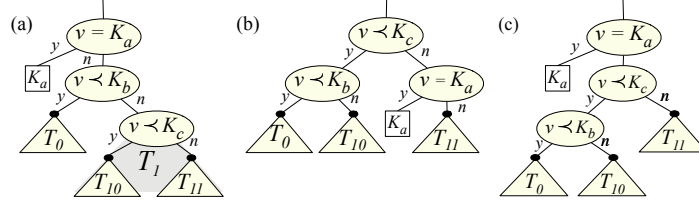


Fig. 4. (a) The subtree T' in Case 2, two possible replacements (b), (c).

As in Fig. 4(a), let the root of T_1 be $\langle v \prec K_c \rangle$, with subtrees T_{l0} and T_{l1} .

Lemma 3. (i) $\omega(T_0) \geq \omega(T_{l1})$. (ii) If $K_a \not\prec K_c$, then $\omega(K_a) \geq \omega(T_1)$.

(As replacing T' by (b) or (c) preserves correctness; proof in Appendix 7.2.)

Case 2.1: $K_a \not\prec K_c$. By Lemma 3(ii), $\omega(K_a) \geq \omega(T_1)$. Descendant $\langle v = K_z \rangle$ is in T_1 , so $\omega(T_1) \geq \omega(K_z)$. Transitivity, $\omega(K_a) \geq \omega(K_z)$, and we are done.

Case 2.2: $K_a \prec K_c$. By Lemma 3(i), $\omega(T_0) \geq \omega(T_{l1})$. By Fact A, $\omega(K_a) \geq \omega(T_{l1})$.

If $\langle v = K_z \rangle$ is in T_{l1} , then $\omega(T_{l1}) \geq \omega(K_z)$ and transitively we are done.

In the remaining case, $\langle v = K_z \rangle$ is in T_{l0} . T 's irreducibility implies $K_z \prec K_c$. Since $K_a \prec K_c$ also (Case 2.2), grandchild $\langle v \prec K_c \rangle$ does not separate K_a from K_z , and by Assumption 1 the root of subtree T_{l0} does an inequality comparison. Hence, the subtree rooted at $\langle v \prec K_b \rangle$ is as in Fig. 5(a):

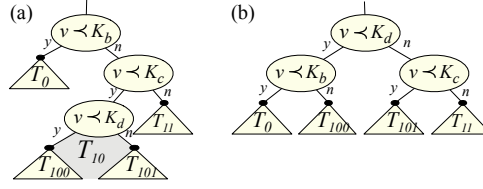


Fig. 5. (a) The subtree rooted at $\langle v \prec K_b \rangle$ in Case 2.2. (b) A possible replacement.

Lemma 4. $\omega(T_0) \geq \omega(T_{10})$.

(Because replacing (a) by (b) preserves correctness; proof in Appendix 7.2.)

Since descendant $\langle v = K_z \rangle$ is in T_{10} , Lemma 4 implies $\omega(T_0) \geq \omega(T_{10}) \geq \omega(K_z)$. This and Fact A imply $\omega(K_a) \geq \omega(K_z)$. This proves Lemma 1. \square

Proposition 1. *If any leaf node $\langle K_\ell \rangle$'s parent P does not do an equality comparison against key K_ℓ , then changing P so that it does so gives an irreducible 2WCST T' of the same cost.*

Proof. Since $\mathcal{Q}_{\langle K_\ell \rangle} = \{K_\ell\}$ and P 's comparison operator is in $\mathcal{C} \subseteq \{<, \leq, =\}$, it must be that $K_\ell = \max \mathcal{Q}_P$ or $K_\ell = \min \mathcal{Q}_P$. So changing P to $\langle v = K_\ell \rangle$ (with $\langle K_\ell \rangle$ as the “yes” child and the other child the “no” child) maintains correctness, cost, and irreducibility. \square

Proof. (Thm. 6) Consider any equality-testing node $N = \langle v = K_a \rangle$ and any key $K_z \in \mathcal{Q}_N$. Since $K_z \in \mathcal{Q}_N$, node N has descendant leaf $\langle K_z \rangle$. Without loss of generality (by Proposition 1), leaf $\langle K_z \rangle$'s parent is $\langle v = K_z \rangle$. That parent is a descendant of $\langle v = K_a \rangle$, so $\omega(K_a) \geq \omega(K_z)$ by Lemma 1. \square

3 Proofs of Thm. 1 (algorithm for 2WCST) and Thm. 3

First we prove Thm. 1. Fix an instance $\mathcal{I} = (\mathcal{K}, \mathcal{Q}, \mathcal{C}, \alpha, \beta)$. Assume for now that all probabilities in β are distinct. For any query subset $\mathcal{S} \subseteq \mathcal{Q}$, let $\text{opt}(\mathcal{S})$ denote the minimum cost of any 2WCST that correctly determines all queries in subset \mathcal{S} (using keys in \mathcal{K} , comparisons in \mathcal{C} , and weights from the appropriate restriction of α and β to \mathcal{S}). Let $\omega(\mathcal{S})$ be the probability that a random query v is in \mathcal{S} . The cost of any tree for \mathcal{S} is the weight of the root ($= \omega(\mathcal{S})$) plus the cost of its two subtrees, yielding the following dynamic-programming recurrence:

Lemma 5. *For any query set $\mathcal{S} \subseteq \mathcal{Q}$ not handled by a single-node tree,*

$$\text{opt}(\mathcal{S}) = \omega(\mathcal{S}) + \min \begin{cases} \min_k \text{opt}(\mathcal{S} \setminus \{k\}) & \text{(if “=” is in } \mathcal{C}, \text{ else } \infty) & (i) \\ \min_{k, \prec} \text{opt}(\mathcal{S}_k^\prec) + \text{opt}(\mathcal{S} \setminus \mathcal{S}_k^\prec), & (ii) \end{cases}$$

where k ranges over \mathcal{K} , and \prec ranges over the allowed inequality operators (if any), and $\mathcal{S}_k^\prec = \{v \in \mathcal{S} : v \prec k\}$.

Using the recurrence naively to compute $\text{opt}(\mathcal{Q})$ yields exponentially many query subsets \mathcal{S} , because of line (i). But, by Thm. 6, we can restrict k in line (i) to be the maximum-likelihood key in \mathcal{S} . With this restriction, the only subsets \mathcal{S} that arise are intervals within \mathcal{Q} , minus some most-likely keys. Formally, for each of $O(n^2)$ key pairs $\{k_1, k_2\} \subseteq \mathcal{K} \cup \{-\infty, \infty\}$ with $k_1 < k_2$, define four *key intervals*

$$\begin{aligned} (k_1, k_2) &= \{v \in \mathcal{Q} : k_1 < v < k_2\}, & [k_1, k_2] &= \{v \in \mathcal{Q} : k_1 \leq v \leq k_2\}, \\ (k_1, k_2] &= \{v \in \mathcal{Q} : k_1 < v \leq k_2\}, & [k_1, k_2) &= \{v \in \mathcal{Q} : k_1 \leq v < k_2\}. \end{aligned}$$

For each of these $O(n^2)$ key intervals I , and each integer $h \leq n$, define $\text{top}(I, h)$ to contain the h keys in I with the h largest β_i 's. Define $\mathcal{S}(I, h) = I \setminus \text{top}(I, h)$. Applying the restricted recurrence to $\mathcal{S}(I, h)$ gives a simpler recurrence:

Lemma 6. *If $\mathcal{S}(I, h)$ is not handled by a one-node tree, then $\text{opt}(\mathcal{S}(I, h))$ equals*

$$\omega(\mathcal{S}(I, h)) + \min \begin{cases} \text{opt}(\mathcal{S}(I, h+1)) & (\text{if equality is in } \mathcal{C}, \text{ else } \infty) & (i) \\ \min_{k, \prec} \text{opt}(\mathcal{S}(I_k^\prec, h_k^\prec)) + \text{opt}(\mathcal{S}(I \setminus I_k^\prec, h - h_k^\prec)), & (ii) \end{cases}$$

where key interval $I_k^\prec = \{v \in I : v \prec k\}$, and $h_k^\prec = |\text{top}(I, h) \cap I_k^\prec|$.

Now, to compute $\text{opt}(\mathcal{Q})$, each query subset that arises is of the form $\mathcal{S}(I, h)$ where I is a key interval and $0 \leq h \leq n$. With care, each of these $O(n^3)$ subproblems can be solved in $O(n)$ time, giving an $O(n^4)$ -time algorithm. In particular, represent each key-interval I by its two endpoints. For each key-interval I and integer $h \leq n$, precompute $\omega(\mathcal{S}(I, h))$, and $\text{top}(I, h)$, and the h 'th largest key in I . Given these $O(n^3)$ values (computed in $O(n^3 \log n)$ time), the recurrence for $\text{opt}(\mathcal{S}(I, h))$ can be evaluated in $O(n)$ time. In particular, for line (ii), one can enumerate all $O(n)$ pairs (k, h_k^\prec) in $O(n)$ time total, and, for each, compute I_k^\prec and $I \setminus I_k^\prec$ in $O(1)$ time. Each base case can be recognized and handled (by a cost-0 leaf) in $O(1)$ time, giving total time $O(n^4)$. This proves Thm. 1 when all probabilities in β are distinct; Sec. 3.1 finishes the proof.

3.1 Perturbation argument; proofs of Theorems 1 and 3

Here we show that, without loss of generality, in looking for an optimal search tree, one can assume that the key probabilities (the β_i 's) are all distinct. Given any instance $\mathcal{I} = (\mathcal{K}, \mathcal{Q}, \mathcal{C}, \alpha, \beta)$, construct instance $\mathcal{I}' = (\mathcal{K}, \mathcal{Q}, \mathcal{C}, \alpha, \beta')$, where $\beta'_j = \beta_j + j\varepsilon$ and ε is a positive infinitesimal (or ε can be understood as a sufficiently small positive rational). To compute (and compare) costs of trees with respect to \mathcal{I}' , maintain the infinitesimal part of each value separately and extend linear arithmetic component-wise in the natural way:

1. Compute $z \times (x_1 + x_2 \varepsilon)$ as $(zx_1) + (zx_2)\varepsilon$, where z, x_1, x_2 are any rationals,
2. compute $(x_1 + \varepsilon x_2) + (y_1 + \varepsilon y_2)$ as $(x_1 + x_2) + (y_1 + y_2)\varepsilon$,
3. and say $x_1 + \varepsilon x_2 < y_1 + \varepsilon y_2$ iff $x_1 < y_1$, or $x_1 = y_1 \wedge x_2 < y_2$.

Lemma 7. *In the instance \mathcal{I}' , all key probabilities β'_i are distinct. If a tree T is optimal w.r.t. \mathcal{I}' , then it is also optimal with respect to \mathcal{I} .*

Proof. Let A be a tree that is optimal w.r.t. \mathcal{I}' . Let B be any other tree, and let the costs of A and B under \mathcal{I}' be, respectively, $a_1 + a_2\varepsilon$ and $b_1 + b_2\varepsilon$. Then their respective costs under \mathcal{I} are a_1 and b_1 . Since A has minimum cost under \mathcal{I}' , $a_1 + a_2\varepsilon \leq b_1 + b_2\varepsilon$. That is, either $a_1 < b_1$, or $a_1 = b_1$ (and $a_2 \leq b_2$). Hence $a_1 \leq b_1$: that is, A costs no more than B w.r.t. \mathcal{I} . Hence A is optimal w.r.t. \mathcal{I} . \square

Doing arithmetic this way increases running time by a constant factor.⁵ This completes the proof of Thm. 1. The reduction can also be used to avoid the significant effort that Anderson et al. [1] devote to non-distinct key probabilities.

For computing optimal *binary split trees* for unrestricted queries, the fastest known time is $O(n^5)$, due to [6]. But [6] also gives an $O(n^4)$ -time algorithm for the case of distinct key probabilities. With the above reduction, the latter algorithm gives $O(n^4)$ time for the general case, proving Thm. 3.

4 Proof of Thm. 2 (additive-3 approximation algorithm)

Fix any instance $\mathcal{I} = (\mathcal{K}, \mathcal{Q}, \mathcal{C}, \alpha, \beta)$. If \mathcal{C} is $\{=\}$ then the optimal tree can be found in $O(n \log n)$ time, so assume otherwise. In particular, $<$ and/or \leq are in \mathcal{C} . Assume that $<$ is in \mathcal{C} (the other case is symmetric).

The entropy $H_{\mathcal{I}} = -\sum_i \beta_i \log_2 \beta_i - \sum_i \alpha_i \log_2 \alpha_i$ is a lower bound on $\text{opt}(\mathcal{I})$. For the case $\mathcal{K} = \mathcal{Q}$ and $\mathcal{C} = \{<\}$, Yeung's $O(n)$ -time algorithm [16] constructs a 2WCST that uses only $<$ -comparisons whose cost is at most $H_{\mathcal{I}} + 2 - \beta_1 - \beta_n$. We reduce the general case to that one, adding roughly one extra comparison.

Construct $\mathcal{I}' = (\mathcal{K}' = \mathcal{K}, \mathcal{Q}' = \mathcal{K}, \mathcal{C}' = \{<\}, \alpha', \beta')$ where each $\alpha'_i = 0$ and each $\beta'_i = \beta_i + \alpha_i$ (except $\beta'_1 = \alpha_0 + \beta_1 + \alpha_1$). Use Yeung's algorithm [16] to construct tree T' for \mathcal{I}' . Tree T' uses only the $<$ operator, so any query $v \in \mathcal{Q}$ that reaches a leaf $\langle K_i \rangle$ in T' must satisfy $K_i \leq v < K_{i+1}$ (or $v < K_2$ if $i = 1$). To distinguish $K_i = v$ from $K_i < v < K_{i+1}$, we need only add one additional comparison at each leaf (except, if $i = 1$, we need two).⁶ By Yeung's guarantee, T' costs at most $H_{\mathcal{I}'} + 2 - \beta'_1 - \beta'_n$. The modifications can be done so as to increase the cost by at most $1 + \alpha_0 + \alpha_1$, so the final tree costs at most $H_{\mathcal{I}'} + 3$. By standard properties of entropy, $H_{\mathcal{I}'} \leq H_{\mathcal{I}} \leq \text{opt}(\mathcal{I})$, proving Thm. 2.

5 Proof of Thm. 4 (errors in work on binary split trees)

A *generalized binary split tree* (GBST) is a rooted binary tree where each node N has an *equality* key e_N and a *split* key s_N . A search for query $v \in \mathcal{Q}$ starts at the root r . If $v = e_r$, the search halts. Otherwise, the search recurses on the left subtree (if $v < s_r$) or the right subtree (if $v \geq s_r$). The *cost* of the tree is the expected number of nodes (including, by convention, leaves) visited for a random query v . Fig. 6 shows two GBSTs for a single instance.

⁵ For an algorithm that works with linear (or $O(1)$ -degree polynomial) functions of β .

⁶ If it is possible to distinguish $v = K_i$ from $K_i < v < K_{i+1}$, then \mathcal{C} must have at least one operator other than $<$, so we can add either $\langle v = K_i \rangle$ or $\langle v \leq K_i \rangle$.

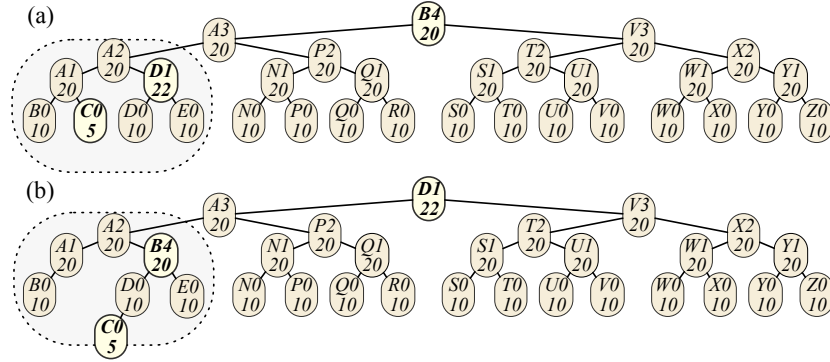


Fig. 6. Two GBSTs for an instance. Keys are ordered alphabetically ($A0 < A1 < A2 < A3 < B0 < \dots$). Each node shows its equality key and the frequency of that key; split keys are not shown. The algorithm of [9] gives (a), of cost 1763, but (b) costs 1762.

To prove Thm. 4, we observe that [9]’s Lemma 4 and algorithm fail on the instance in Fig. 6. There is a solution of cost only 1762 (in Fig. 6(b)), but the algorithm gives cost 1763 for the instance (as in Fig. 6(a)), as can be verified by executing the Python code for the algorithm in Appendix 7.1. The intuition is that the optimal substructure property fails for the subproblems defined by [9]: the circled subtree in (a) (with root $A2$) is cheaper than the corresponding subtree in (b), but leads to larger global cost. For more intuition and the full proof, see Appendix 7.3.

6 Proof of Thm. 5 ($O(n \log n)$ time without equality)

Fix any 2WCST instance $\mathcal{I} = (\mathcal{K}, \mathcal{Q}, \mathcal{C}, \alpha, \beta)$ with $\mathcal{C} \subseteq \{<, \leq\}$. Let $n = |\mathcal{K}|$. We show that, in $O(n \log n)$ time, one can compute an equivalent instance $\mathcal{I}' = (\mathcal{K}', \mathcal{Q}', \mathcal{C}', \alpha', \beta')$ with $\mathcal{K}' = \mathcal{Q}'$, $\mathcal{C}' = \{<\}$, and $|\mathcal{K}'| \leq 2n + 1$. (*Equivalent* means that, given an optimal 2WCST T' for \mathcal{I}' , one can compute in $O(n \log n)$ time an optimal 2WCST T for \mathcal{I} .) The idea is that, when $\mathcal{C} \subseteq \{<, \leq\}$, open intervals are functionally equivalent to keys.

Assume without loss of generality that $\mathcal{C} = \{<, \leq\}$. (Otherwise no correct tree exists unless $\mathcal{K} = \mathcal{Q}$, and we are done.) Assume without loss of generality that no two elements in \mathcal{Q} are equivalent (in that they relate to all keys in \mathcal{K} in the same way; otherwise, remove all but one query from each equivalence class). Hence, at most one query lies between any two consecutive keys, and $|\mathcal{Q}| \leq 2|\mathcal{K}| + 1$.

Let instance $\mathcal{I}' = (\mathcal{K}', \mathcal{Q}, \mathcal{C}', \alpha', \beta')$ be obtained by taking the key set $\mathcal{K}' = \mathcal{Q}$ to be the key set, but restricting comparisons to $\mathcal{C}' = \{<\}$ (and adjusting the probability distribution appropriately — take $\alpha' \equiv 0$, take β_i to be the probability associated with the i th query — the appropriate α_j or β_j).

Given any irreducible 2WCST T for \mathcal{I} , one can construct a tree T' for \mathcal{I}' of the same cost as follows. Replace each node $\langle v \leq k \rangle$ with a node $\langle v < q \rangle$, where q is the least query value larger than k (there must be one, since $\langle v \leq k \rangle$ is in T and T is irreducible). Likewise, replace each node $\langle v < k \rangle$ with a node $\langle v < q \rangle$, where q is the least query value greater than or equal to k (there must be one, since $\langle v < k \rangle$ is in T and T is irreducible). T' is correct because T is.

Conversely, given any irreducible 2WCST T' for \mathcal{I}' , one can construct an equivalent 2WCST T for \mathcal{I} as follows. Replace each node $N' = \langle v < q \rangle$ as follows. If $q \in \mathcal{K}$, replace N' by $\langle v < k \rangle$. Otherwise, replace N' by $\langle v \leq k \rangle$, where key k is the largest key less than q . (There must be such a key k . Node $\langle v < q \rangle$ is in T' but T' is irreducible, so there is a query, and hence a key k , smaller than q .) Since T' correctly classifies each query in \mathcal{Q} , so does T .

To finish, we note that the instance \mathcal{I}' can be computed from \mathcal{I} in $O(n \log n)$ time (by sorting the keys, under reasonable assumptions about \mathcal{Q}), and the second mapping (from T' to T) can be computed in $O(n \log n)$ time. Since \mathcal{I}' has $\mathcal{K}' = \mathcal{Q}'$ and $\mathcal{C} = \{<\}$, it is known [10] to be equivalent to an instance of alphabetic encoding, which can be solved in $O(n \log n)$ time [7,4].

References

1. R. Anderson, S. Kannan, H. Karloff, and R. E. Ladner. Thresholds and optimal binary comparison search trees. *Journal of Algorithms*, 44:338–358, 2002.
2. M. Chrobak, M. Golin, J. I. Munro, and N. E. Young. Optimal search trees with 2-way comparisons. In *Proceedings of International Symposium on Algorithms and Computation*, 2015.
3. D. Comer. A note on median split trees. *ACM Transactions on Programming Languages and Systems*, 2(1):129–133, 1980.
4. A. M. Garsia and M. L. Wachs. A new algorithm for minimum cost binary trees. *SIAM Journal on Computing*, 6(4):622–642, 1977.
5. E. Gilbert and E. Moore. Variable-length binary encodings. *Bell System Technical Journal*, 38(4):933–967, 1959.
6. J. H. Hester, D. S. Hirschberg, S. H. Huang, and C. K. Wong. Faster construction of optimal binary split trees. *Journal of Algorithms*, 7(3):412–424, 1986.
7. T. C. Hu and A. C. Tucker. Optimal computer search trees and variable-length alphabetical codes. *SIAM Journal on Applied Mathematics*, 21(4):514–532, 1971.
8. S. Huang and C. Wong. Optimal binary split trees. *Journal of Algorithms*, 5(1):69–79, 1984.
9. S.-H. Huang and C. K. Wong. Generalized binary split trees. *Acta Informatica*, 21(1):113–123, 1984.
10. D. E. Knuth. Optimum binary search trees. *Acta Informatica*, 1(1):14–25, 1971.
11. D. E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley Publishing Company, 2nd edition, 1998.
12. Y. Perl. Optimum split trees. *Journal of Algorithms*, 5:367–374, 1984.
13. B. A. Sheil. Median split trees: a fast lookup technique for frequently occurring keys. *Communications of the ACM*, 21(11):947–958, 1978.
14. D. Spuler. Optimal search trees using two-way key comparisons. *Acta Informatica*, 24:729–740, 1994.

15. D. A. Spuler. *Optimal Search Trees Using Two-Way Key Comparisons*. PhD thesis, James Cook University, 1994.
16. R. Yeung. Alphabetic codes revisited. *IEEE Transactions on Information Theory*, 37(3):564–572, 1991.

7 Appendix

7.1 Python code for Thm. 4 (GBST algorithm of [9])

```

1  #!/usr/bin/env python3.4
2  import functools
3  memoize = functools.lru_cache(maxsize=None)
4
5  def huang1984(weights):
6      "Returns cost as computed by Huang and Wong's GBST algorithm (1984)."
```

```

7
8      n = len(weights)
9      beta = {i+1 : weights[key] for i, key in enumerate(sorted(weights.keys()))}
10
11      def is_legal(i, j, d): return 0 <= i <= j <= n and 0 <= d <= j - i
12
13      @memoize
14      def p_w_t(i, j, d):
15          "Returns triple: (cost p[i,j,d], weight w[i,j,d], deleted keys for t[i,j,d])."
```

```

16
17          interval = set(range(i+1, j+1))
18
19          if d == j-i: # base case
20              return (0, 0, interval)
21
22          def candidates(): # Lemma 4 recurrence from Huang et al
23              for k in interval: # k = index of split key
24                  for m in range(d+2): # m = num. deletions from left subtree
25                      if is_legal(i, k-1, m) and is_legal(k-1, j, d-m+1):
26                          cost_l, weight_l, deleted_l = p_w_t(i, k-1, m)
27                          cost_r, weight_r, deleted_r = p_w_t(k-1, j, d-m+1)
28                          deleted = deleted_l.union(deleted_r)
29                          x = min(deleted, key = lambda h : beta[h])
30                          weight = beta[x] + weight_l + weight_r
31                          cost = weight + cost_l + cost_r
32                          yield cost, weight, deleted -set([x])
33
34          return min(candidates())
35
36      cost, weight, keys = p_w_t(0, n, 0)
37      return cost
38
39  weights = dict(b4=20,
40                a3=20, v3=20,
41                a2=20, p2=20, t2=20, x2=20,
42                a1=20, d1=22, n1=20, q1=20, s1=20, u1=20, w1=20, y1=20,
43                b0=10, c0= 5, d0=10, e0=10, n0=10, p0=10, q0=10, r0=10,
44                s0=10, t0=10, u0=10, v0=10, w0=10, x0=10, y0=10, z0=10)
45
46  assert huang1984(weights) == 1763 # Both assertions pass. The first is used in our Thm. 4.
47
48  weights['d1'] += 0.99 # Increasing a weight cannot decrease the optimal cost, but
49  assert huang1984(weights) < 1763 # in this case decreases the cost computed by the algorithm.
```

The extended abstract [2] is essentially the body of this paper minus the remainder of this appendix. The remainder of this appendix contains all proofs omitted from the extended abstract.

7.2 Proof of Lemmas 2–4 (in the proof of Spuler’s conjecture)

We prove some slightly stronger lemmas that imply Lemmas 2–4.

Let T be any irreducible, optimal 2WCST as in the proof of Lemma 1.

Lemma 8 (implies Lemma 2). *Assume T has a subtree as in Fig. 3(a) with nodes $\langle v = K_a \rangle$ and $\langle v \prec K_b \rangle$. (i) Replacing that subtree the one in Fig. 3(b) (if $K_a \prec K_b$) or the one in Fig. 3(c) (if $K_a \not\prec K_b$) preserves correctness. (ii) If $K_a \prec K_b$, then $\omega(K_a) \geq \omega(T_1)$; otherwise $\omega(K_a) \geq \omega(T_0)$.*

Proof. Assume that $K_a \prec K_b$ (the other case is symmetric). By inspection of each case ($Q = K_a$ or $Q \neq K_a$), subtree (b) classifies each query Q the same way subtree (a) does, so the modified tree is correct. The modification changes the cost by $\omega(K_a) - \omega(T_1)$, so (since T has minimum cost) $\omega(K_a) \geq \omega(T_1)$. \square

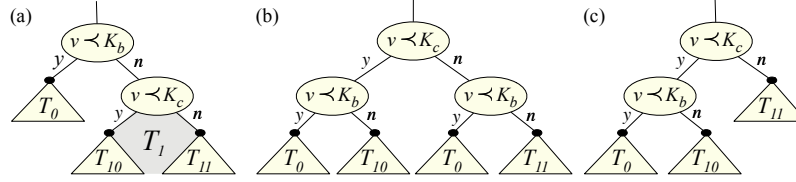


Fig. 7. Lemma 9 — “Rotating” subtree (a) yields (c); the subtrees are interchangeable.

Lemma 9 (implies Lemma 3(i)). (i) *If T has either of the two subtrees in Fig. 7(a) or (c), then exchanging one for the other preserves correctness. (ii) If T has the subtree in Fig. 7(a), then $\omega(T_0) \geq \omega(T_{11})$.*

Proof. Part (i). The transformation from (a) to (c) is a standard rotation operation on binary search trees, but, since the comparison operators can be either $<$ or \leq in our context, we verify correctness carefully.

By inspection, replacing subtree (a) by subtree (b) (in Fig. 7) gives a tree that classifies all queries as T does, and so is correct.

Next we observe that, in subtree (b), replacing the right subtree by just T_{11} (to obtain subtree (c)), maintains correctness. Indeed, since T is irreducible, replacing (in (a)) the subtree T_1 by just T_{11} would give an incorrect tree. Equivalently, $\exists Q. Q \not\prec K_b \wedge Q \prec K_c$. Equivalently, the right-bounded interval $\{Q \in \mathbb{R} : Q \prec K_c\}$ overlaps the left-bounded interval $\{Q \in \mathbb{R} : Q \not\prec K_b\}$. Equivalently, the complements of these intervals, namely $\{Q \in \mathbb{R} : Q \not\prec K_c\}$ and $\{Q \in \mathbb{R} : Q \prec K_b\}$, are disjoint. Equivalently, $\forall Q. Q \not\prec K_c \rightarrow Q \not\prec K_b$. Hence, replacing the right subtree of (b) by T_{11} (yielding (c)) maintains correctness.

In sum, replacing subtree (a) by subtree (c) maintains correctness. This shows part (i). This replacement changes the cost by $\omega(T_0) - \omega(T_{11})$, so $\omega(T_0) \geq \omega(T_{11})$. This proves part (iii). The proof of (ii) is symmetric to the proof of (i). \square

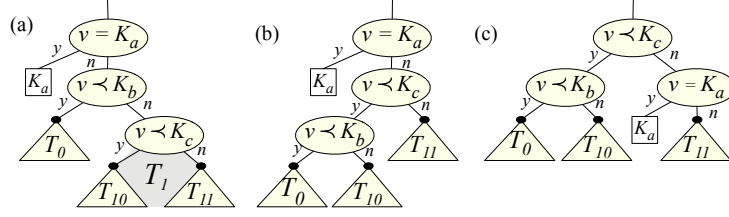


Fig. 8. Lemma 10 — Subtrees (a) and (c) are interchangeable if $K_a \not\prec K_c$.

Lemma 10 (implies Lemma 3(ii)). *If T has a subtree as in Fig. 8(a), and $K_a \not\prec K_c$, then (i) replacing the subtree by Fig. 8(c) preserves correctness, and (ii) $\omega(K_a) \geq \omega(T_1)$.*

Proof. (i) Assume T has the subtree in Fig. 7(a) (the other case is symmetric). By Lemma 9(i) (applied to the subtree of (a) with root $\langle v \prec K_b \rangle$), replacing subtree (a) by subtree (b) gives a correct tree. Then, by Lemma 8(i) (applied to subtree (b), but note that node $\langle v \prec K_c \rangle$ in (b) takes the role of node $\langle v \prec K_b \rangle$ in Fig. 3(a)!) replacing (b) by (c) gives a correct tree. This proves part (i). Part (ii) follows because replacing (a) by (c) changes the cost of T by $\omega(K_a) - \omega(T_1)$, and T has minimum cost, so $\omega(K_a) \geq \omega(T_1)$. \square

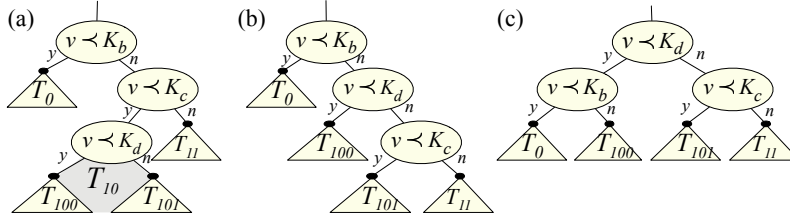


Fig. 9. Lemma 11 — Subtrees (a) and (c) are interchangeable.

Lemma 11 (implies Lemma 4). *If T has a subtree as in Fig. 9(a), then (i) replacing that subtree by the one in Fig. 9(c) preserves correctness, and (ii) $\omega(T_0) \geq \omega(T_{10})$.*

Proof. Applying Lemma 9(i) to the subtree of (a) with root $\langle v \prec K_c \rangle$, replacing subtree (a) by subtree (b) gives a correct tree.⁷ Then, applying Lemma 9(i) to the subtree of (b) with root $\langle v \prec K_b \rangle$, replacing subtree (b) by subtree (c) gives

⁷ Technically, to apply Lemma 9(i), we need (b) to be correct and *irreducible*. The overall argument remains valid though as long as the tree from Fig. 9(a) is optimal.

a correct tree. This shows part (i). Part (ii) follows, because replacing (a) by (c) changes the cost of T by $\omega(T_0) - \omega(T_{10})$, so $\omega(T_0) \geq \omega(T_{10})$. \square

7.3 Proof of Thm. 4 — Huang and Wong’s error

A *generalized binary split tree* (GBST) is a rooted binary tree where each node N has an *equality* key e_N and a *split* key s_N . A search for query $v \in \mathcal{Q}$ starts at the root r . If $v = e_r$, the search halts. Otherwise, the search recurses on the left subtree (if $v < s_r$) or the right subtree (if $v \geq s_r$). The *cost* of the tree is the expected number of nodes (including, by convention, leaves) visited for a random query v .

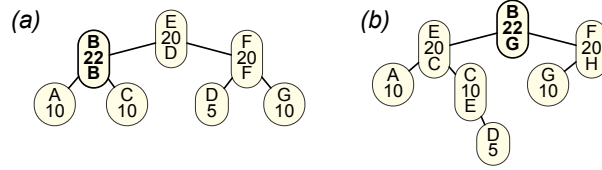


Fig. 10. In a GBST, each node has equality key, frequency, and (if internal) split key.

Huang and Wong demonstrate that equality keys in optimal GBSTs do not have the maximum-likelihood property [9]. Fig. 10 shows their counterexample: in the optimal GBST (a), the root equality key is E (frequency 20), not B (frequency 22). The cheapest tree with B at the root is (b), and is more expensive. Having B at the root increases the cost because then the other two high-frequency keys E and F have to be the children, so the split key of the root has to split E and F, and low-frequency keys A, C, and D all must be in the left subtree.

Following [9], restrict to successful queries ($\mathcal{K} = \mathcal{Q}$). Fix any instance $\mathcal{I} = (\mathcal{K}, \beta)$. For any query interval $I = \{K_i, K_{i+1}, \dots, K_j\}$ and any subset $D \subseteq I$ of “deleted” keys, let $\text{opt}(I, D)$ denote the minimum cost of any GBST that handles the keys in $I \setminus D$. This recurrence follows directly from the definition of GBSTs:

Lemma 12. *For any query set $I \setminus D$ not handled by a single-node tree,*

$$\text{opt}(I, D) = \omega(I \setminus D) + \min_{e, s \in \mathcal{K}} \text{opt}(I_{<s}, D_e \cap I_{<s}) + \text{opt}(I_{\geq s}, D_e \cap I_{\geq s})$$

where $D_e = D \setminus \{e\}$, and $I_{<s} = \{v \in I : v < s\}$ and $I_{\geq s} = \{v \in I : v \geq s\}$.

The goal is to compute $\text{opt}(\mathcal{K}, \emptyset)$. Using the recurrence above, exponentially many subsets D arise. This motivates the following lemma. For any node N in an optimal GBST, define N ’s *key interval*, I_N , and *deleted-key set*, D_N , according to the recurrence in the natural way. Then the set \mathcal{Q}_N of queries reaching N is $I_N \setminus D_N$, and D_N contains those keys in I_N that are in equality tests above N , and I_N contains the key values that, if searched for in T with the equality tests removed, would reach node N .

Lemma 13 ([9, Lemma 2]). *For any node N in an optimal GBST, N 's equality key is a least-frequent key among those in I_N that aren't equality keys in any of N 's subtrees: if $e_N = K_i$, then $\beta_i = \min\{\beta_j : K_j \in D_N\}$.*

The proof is the same exchange argument that shows our Assumption 1(ii).

[9] claims (incorrectly) that, by Lemma 13, the desired value $\text{opt}(\mathcal{Q}, \emptyset)$ can be computed as follows. For any key interval $I = \{K_{i+1}, \dots, K_j\}$ and $d \leq n$, let

$$p[i, j, d] = \min\{\text{opt}(I \setminus D) : D \subseteq I, |D| = d\} \quad (1)$$

be the minimum cost of any GBST for any query set $I \setminus D$ consisting of I minus any d deleted keys. Let $t[i, j, d]$ be a corresponding subtree of cost $p[i, j, d]$, and let $w[i, j, d]$ be the weight of the root of $t[i, j, d]$.

Their algorithm uses the following (incorrect) recurrence (their Lemma 4):

$$p[i, j, d] = \min(w[i, j, d] + p[i, k-1, m] + p[k-1, j, d-m-1])$$

where the minimum is taken over all legal combinations of k 's and m 's [and]

$$w[i, j, d] = w[i, k-1, m] + w[k-1, j, d-m-1] + \beta_x$$

where x is the index of the key of minimum frequency among those in range $\{K_{i+1}, \dots, K_j\}$ but outside $t[i, k-1, m]$ and $t[k-1, j, d-m-1]$.

Next we describe their error. Recall that $p[i, j, d]$ chooses a subtree of *minimum cost* (among trees with any d keys deleted). But this choice might not lead to minimum overall cost! The reason is that the subtree's cost does not suffice to determine the contribution to the overall cost: the *weight* of the subtree, and the weights of the deleted keys and their eventual locations, also matter.

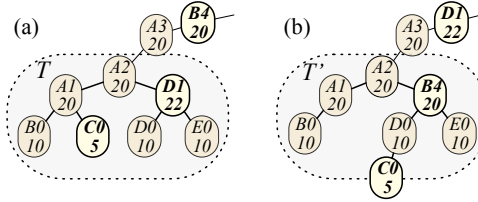


Fig. 11. Trees T, T' for 9-key interval I with $d = 2$. (Split keys not shown.)

For example, consider $p[i, j, d]$ for the subproblem where $d = 2$ and interval I consists of the nine keys $I = \{A1, A2, A3, B0, B4, C0, D0, D1, E0\}$, with the following weights:

key:	A1	A2	A3	B0	B4	C0	D0	D1	E0
weight:	20	20	20	10	20	5	10	22	10

Fig. 11 shows two 7-node subtrees (circled and shaded), called T and T' , involving these keys. These subtrees will be used in our counter-example, described below. (The split key of each node is not shown in the diagram.)

Partition the set of possible trees $t[i, j, d]$ into two classes: (i) those that contain $D1$ and (ii) those that don't (that is, $D1$ is a “deleted” key). By a careful case analysis,⁸ subtree T in Fig. 11(a) is a cheapest (although not unique) tree in class (i), while the 7-node subtree T' in Fig. 11(b) is a cheapest tree in class (ii). Further, the subtree T' costs 1 more than the subtree T . Hence, the algorithm of [9] will choose T , not T' , for this subproblem.

However, this choice is incorrect. Consider not just the cost of tree, but also the effects of the choice on the deleted keys' costs. For definiteness, suppose the two deleted nodes become, respectively, the parent and grandparent of the root of the subtree, as in (a) and (b) of the figure. In (b), $C0$ is one level deeper than it is in (a), which increases the cost by 5, but $D1$ is three levels higher, which decreases the overall cost by $2 \times 3 = 6$, for a net decrease of 1 unit. Hence, using T instead of T' ends up costing the overall solution 1 unit more.

This observation is the basis of the complete counterexample shown in Fig. 6. The counterexample extends the smaller example above by appending two “neutral” subintervals, with 7 and 15 keys, respectively, each of which (without any deletions) admits a self-contained balanced tree. Keys are ordered alphabetically. On this instance, the algorithm of [9] (and their Lemma 4) fail, as they choose T instead of T' for the subproblem. Fig. 6(a) shows the tree computed by their recurrence, of cost 1763. (This can be verified by executing the Python code for the algorithm in Appendix 7.1.) Fig. 6(b) shows a tree that costs 1 less. This proves Thm. 4.

Spuler's thesis. In addition to Spuler's conjecture (and 2WCST algorithms that rely on his conjecture), Spuler's thesis [15, §6.4.1 Conj.1] also presents code for two additional algorithms that he claims, without proof, compute optimal 2WCSTs independently of his conjecture.

First, [15, §6.4.1] gives code for the problem restricted to successful queries ($\mathcal{Q} = \mathcal{K}$), which it describes as a “straightforward” modification of Huang et al.'s algorithm [9] for generalized split trees (an algorithm that we now know is incorrect, per our Thm. 4). Correctness is addressed only by the remark that “*The changes to the optimal generalized binary split tree algorithm of Huang and Wong [9] to produce optimal generalized two-way comparison trees*⁹*are quite straight forward.*”

Secondly, [15, §6.5] gives code for the case of unrestricted queries, which it describes as a “not difficult” modification of the preceding algorithm in §6.4.1. Correctness is addressed only by the following remark: “*The algorithm of the previous section assumes that $\alpha_i = 0$ for all i . However, the improvement of*

⁸ In case (i), to minimize cost, the top two levels of the tree must contain $D1$ and two other heavy keys from $\{A1, A2, A3, B4\}$. $D1$ has to be the right child of the root, because otherwise a key no larger than $B4$ is, so the split key at the root has to be no larger than $B4$, so the three light keys $\{C0, D0, E0\}$ have to all be in the right subtree, so that one of them has to be at level four instead of level three, increasing the cost by at least 5. In case (ii), when $D1$ is deleted (not in the subtree), by a similar argument, one of the light keys has to be at level four, so T' is best possible.

⁹ Spuler's *generalized two-way comparison trees* are exactly 2WCSTs as defined herein.

the algorithm to allow non-zero values of α is not difficult.” In contrast, Huang et al. explicitly mention that they were unable to generalize their algorithm to unrestricted queries [9].

Neither of Spuler’s two proposed algorithms is published in a peer-reviewed venue (although they are referred to in [14]). They have no correctness proofs, and are based on [9], which we now know is incorrect. Given these considerations, we judge that Anderson et al. [1] give the first correct proof of a poly-time algorithm to find optimal 2WCSTs when only successful queries are allowed ($\mathcal{K} = \mathcal{Q}$), and that this paper gives the first correct proof for the general case.